TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE ORIZABA

Asignatura: Estructura de Datos

Carrera: Ingeniería en Informática

Estudiantes:

- Hernandez Angón Diego - No. Control 21010193
- Dorantes Rodríguez Diego Yael - No. Control 21010184

Profesora: María Jacinta Martínez Castillo

Grupo: 3a3A

Fecha de entrega: 22/04/2023

Reporte de unidad 2: Recursividad

INTRODUCCION

El siguiente reporte realizado para la asignatura de Estructura de datos, en la carrera de ingeniería informática, correspondiente al tema recursividad de la unidad 2, tendrá como finalidad realizar un resumen detallado de los conocimientos y experiencias adquiridas a lo largo de la unidad. Se utilizarán diversas fuentes de conocimiento y programas realizados en clase para el siguiente reporte, tomando en cuenta a la iteración, la cual consiste en la repetición de un segmento de código dentro de un programa. Asi como también se utilizará la recursividad, la cual nos permite, de igual forma, que un bloque de código se ejecute un cierto número de veces en un código de programación. La iteración y la recursividad pueden parecer idénticas, no obstante, cada una tiene sus propias características que las hacen diferentes. Se utilizará el lenguaje de programación de Java, y eclipse como IDE de programación.

COMPETENCIA ESPECIFICA

Desarrollar habilidad para la creación de código en java, utilizando la recursividad como principal recurso, identificando cada una de las características y el claro concepto de la recursividad. Ser capaz de diferenciar entre un ejercicio iterativo y un ejercicio recursivo, y aplicar los conocimientos adquiridos en futuros proyectos de programación.

MARCO TEORICO

La recursividad es una técnica de programación que permite que una función se llame a sí misma para resolver un problema de manera más eficiente. En la recursividad, un problema se divide en subproblemas más pequeños y se resuelve recursivamente cada uno de ellos hasta llegar a una solución. Es importante tener en cuenta que la recursividad puede ser muy útil para resolver problemas de manera elegante y eficiente, pero también puede ser peligrosa si no se implementa correctamente. Si una función recursiva se llama a sí misma demasiadas veces, puede generar un desbordamiento de la pila y hacer que el programa falle. Por lo

tanto, se debe tener cuidado al utilizar la recursividad y asegurarse de que siempre haya una condición de salida que garantice que la función terminará en algún momento. Para crear un programa recursivo se debe definir la función recursiva: Se define una función que se llamará a sí misma, definir la condición de salida: Se establece una condición que indica cuándo la función debe dejar de llamarse a sí misma y devolver un resultado. Y, por último, definir la llamada recursiva: Se llama a la función recursiva dentro de sí misma para resolver el problema de manera más eficiente.

Un programa iterativo es aquel que utiliza un bucle o ciclo para repetir un conjunto de instrucciones un número determinado de veces. En lugar de escribir la misma secuencia de instrucciones varias veces en un programa, el bucle permite que estas instrucciones se ejecuten repetidamente, hasta que se cumpla una condición de salida.

El bucle en un programa iterativo puede ser controlado por una variable que se incrementa o decrementa en cada iteración, o por una condición booleana que se evalúa en cada iteración. Los bucles más comunes en la programación son el bucle for, el bucle while y el bucle do-while.

ANALISIS DE ALGORITMOS

El análisis de algoritmos es una rama fundamental de la informática que se centra en la evaluación de la eficiencia de los algoritmos y en la predicción del tiempo de ejecución y la utilización de recursos de los mismos.

Los algoritmos son procedimientos bien definidos que se utilizan para resolver problemas en la informática y en muchas otras áreas. El análisis de algoritmos se utiliza para comparar diferentes algoritmos que resuelven el mismo problema y determinar cuál es más eficiente en términos de tiempo y espacio.

El análisis de algoritmos es importante porque permite a los desarrolladores de software tomar decisiones informadas sobre qué algoritmos utilizar en diferentes situaciones. Por ejemplo, si se necesita un algoritmo para ordenar una lista de elementos, el análisis de algoritmos puede ayudar a determinar cuál de los muchos algoritmos de ordenación disponibles es el más adecuado para la tarea en cuestión.

En conclusión, el análisis de algoritmos es una técnica fundamental para la informática y otras áreas relacionadas que involucran la resolución de problemas mediante el uso de algoritmos. La comprensión de la eficiencia de los algoritmos es crucial para tomar decisiones informadas en el desarrollo de software y en otras aplicaciones donde los algoritmos son utilizados para resolver problemas.

COMPLEJIDAD EL TIEMPO

La complejidad en el tiempo es una medida de la cantidad de tiempo que tarda un algoritmo en resolver un problema. Se utiliza para evaluar la eficiencia de un algoritmo y determinar su capacidad para manejar grandes conjuntos de datos.

El análisis de la complejidad en el tiempo es importante para el desarrollo de software porque ayuda a los programadores a elegir los algoritmos más adecuados para un problema determinado. Un algoritmo más eficiente en términos de tiempo puede procesar grandes cantidades de datos en un tiempo razonable, lo que es crucial en aplicaciones como el procesamiento de imágenes, la simulación de sistemas complejos y la minería de datos.

Una de las tecnicas utilizadas para analizar la complejidad en el tiempo es el análisis de la complejidad media, que se utiliza para evaluar el rendimiento del algoritmo en una variedad de entradas aleatorias, y el análisis de la complejidad en el mejor caso, que se utiliza para evaluar el rendimiento del algoritmo en el mejor de los casos.

En conclusion, la complejidad en el tiempo es una medida importante en programación que se utiliza para evaluar la eficiencia de los algoritmos. Los programadores pueden utilizar el análisis de la complejidad en el tiempo para elegir los algoritmos más adecuados para un problema determinado y garantizar que su software sea capaz de procesar grandes cantidades de datos en un tiempo razonable.

COMPLEJIDAD EN EL ESPACIO

La misma idea que se utiliza para medir la complejidad en tiempo de un algoritmo se utiliza para medir su complejidad en espacio. Decir que un programa es O(n) en espacio significa que sus requerimientos de memoria aumentan proporcionalmente

con el tamaño del problema. Esto es, si el problema se duplica, se necesita el doble de memoria. Del mismo modo, para un programa de complejidad O(n^2) en espacio, la cantidad de memoria que se necesita para almacenar los datos crece con el cuadrado del tamaño del problema: si el problema se duplica, se requiere cuatro veces más memoria. En general, el cálculo de la complejidad en espacio de un algoritmo es un proceso sencillo que se realiza mediante el estudio de las estructuras de datos y su relación con el tamaño del problema.

El análisis de los requerimientos dinámicos de memoria es relativo a los lenguajes que proveen mecanismos de asignación dinámica de la misma. En este caso, la complejidad en espacio viene dada por la cantidad de objetos existentes en un punto del programa, según las reglas de alcance. Así, la cantidad de espacio requerido no será la sumatoria de todas las declaraciones de datos, sino la máxima cantidad de memoria en uso en un momento dado de la ejecución del programa.

EFICIENCIA DE LOS ALGORITMOS

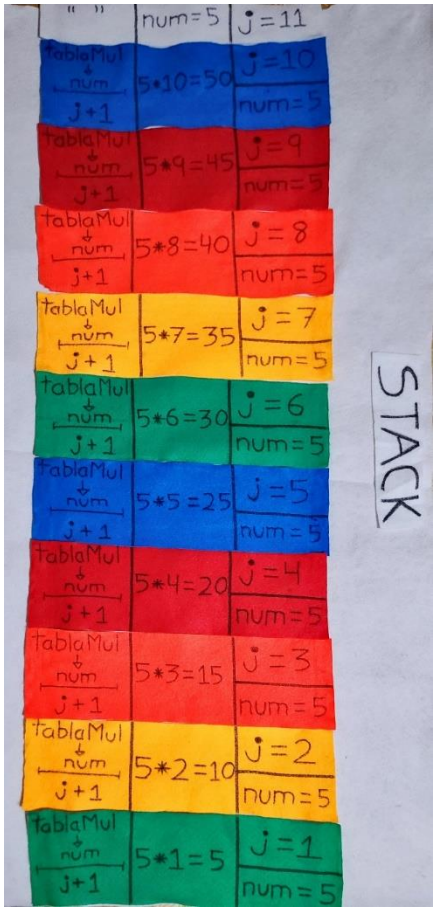
La eficiencia de los algoritmos es un tema importante en la ciencia de la computación, ya que se refiere a la capacidad de los algoritmos para procesar grandes cantidades de datos en un tiempo razonable. La eficiencia de los algoritmos se mide típicamente en términos de tiempo y espacio.

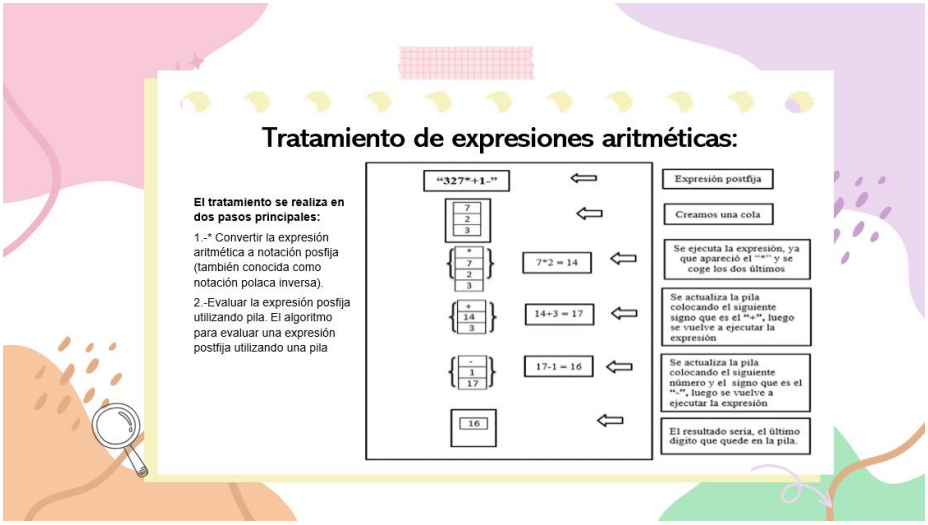
El tiempo de ejecución de un algoritmo se refiere a la cantidad de tiempo que tarda en completar una tarea determinada, y se mide generalmente en términos de la cantidad de operaciones o pasos que requiere el algoritmo. El espacio de un algoritmo se refiere a la cantidad de memoria que requiere para ejecutarse, y se mide generalmente en términos de la cantidad de almacenamiento necesario para los datos y variables que utiliza el algoritmo.

La eficiencia de los algoritmos es importante en muchas áreas de la informática, incluyendo la inteligencia artificial, la minería de datos, la programación de sistemas, la criptografía y la optimización. Un algoritmo eficiente puede ser la diferencia entre un programa que puede procesar grandes cantidades de datos en un tiempo razonable y uno que tarda horas o incluso días en completar una tarea.

En conclusión, la eficiencia de los algoritmos es un tema crítico en la ciencia de la computación. La capacidad de un algoritmo para procesar grandes cantidades de datos en un tiempo razonable puede tener un gran impacto en el rendimiento y la eficacia de una aplicación o sistema informático. Por lo tanto, es importante que los desarrolladores de software tengan en cuenta la eficiencia de los algoritmos al diseñar y optimizar sus programas.

Evidencia- Diego Hernandez Angón





TECNOLÓGICO NACIONAL DE MÉXICO

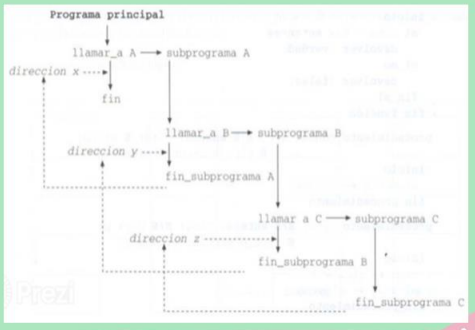
Llamadas a subprogramas

Cuando dentro de un programa se realizan llamadas a subprogramas (o funciones) se utilizan pilas para gestionar la ejecución de dichas llamadas. El programa principal debe recordar el lugar donde se hizo la llamada, de modo que pueda retornar allí cuando el subprograma se haya terminado de ejecutar.

Esta operación se consigue disponiendo las direcciones de retorno de una pila.

Explicacion

- Supongamos que tenemos 3 subprogramas llamados A, B y C.
- Supongamos que A invoca a B y B invoca a C.
- B no terminara su trabajo hasta que C haya terminado y devuelva su control a B
- A es el primero que arranca la ejecución, pero es el ultimo que lo termina, tras la terminación y retorno de B.



Recursividad

Técnica que permite a un bloque de instrucciones que se pueda ejecutar un cierto número de veces.

Para simular un programa resurtivo es necesario la utilización de pila, ya que se está llamando continuamente a sí mismo.

La implementación de una función recursiva en Java sigue los mismos principios que cualquier otra función. La función recursiva tiene que tener un caso base que indica cuándo la recursividad debe detenerse.

```
public static String tablaMul(int num, int j)
{
    if(j<=10)
    {
        return +num+"*"+j+"="+(num*j)+"\n"+tablaMul(num,(j+1));
    }
    else
        return "";
}
```

una pila es una colección homogénea de datos en la que el acceso se realiza siguiendo un criterio LIFO (Last In First Out).

Las operaciones que lleva a cabo son:

- Push. se añade un elemento a la pila
- Pop. se elimina el elemento frontal de la pila
- Free. vacia la pila
- IsEmpty. devuelve cierto si la pila está sin elementos o falso en caso de que contenga uno
- Peek. devuelve el elemento que esta en la cima de la pila.

¡IMPORTANTE!

Ordenación rápida:

Para implementar una organización rápida en pilas, se pueden seguir los siguientes pasos:

1.-Identificar todas las tareas que necesitan ser completadas y escribirlas en notas adhesivas, tarjetas o algún otro medio físico.

2.-Organizar las tareas en orden de importancia o prioridad, colocando las tareas más importantes en la parte superior de la pila

.3.-Tomar la tarea en la parte superior de la pila y trabajar en ella hasta que esté completa.

4.-Una vez que se completa la tarea, moverla a un área de "tareas completadas" o descartarla si no es necesario conservar un registro.

5.-Tomar la siguiente tarea en la parte superior de la pila y continuar trabajando de esta manera hasta que todas las tareas estén completadas.. Esta técnica se puede utilizar en combinación con otras herramientas de gestión de proyectos y tareas, como listas de verificación, calendarios y herramientas de seguimiento del tiempo. La organización rápida en pilas puede ser especialmente útil para aquellos que tienen múltiples tareas y proyectos que deben completarse en un plazo ajustado.

MATERIAL Y EQUIPO

El proyecto se lleva a cabo con los debidos requisitos necesarios. Contar con una computadora con los componentes adecuados, capaz de soportar IDES de programación con lenguaje de java. Se necesita de software específico para la creación de código en Java, el cual puede ser Eclipse o NetBeans. Se cuenta además con los conocimientos otorgados por la profesora en clase, además de material de apoyo para reforzar el conocimiento

DESARROLLO Y RESULTADOS

Para el siguiente reporte, se utilizarán 2 programas, cada uno con su respectivo código en iterativo como en recursivo. Para llevar a cabo la practica fueron necesarios los requisitos anteriormente mencionados. Se toma en cuenta la definición de un ejercicio recursivo y su diferencia con los ejercicios iterativos, recordando que en la recursividad un método se llama asi mismo. Se utilizaron plantillas de métodos anteriormente creados para la entrada y salida de datos, esto para una eficiencia y organización de los códigos, además, se crea una clase principal a donde serán invocados los métodos para un mejor control de la practica

.

```
public static void ImprimeMsje(String msje){
    JOptionPane.showMessageDialog(null,msje,"Salida",JOptionPane.INFORMATION_MESSAGE);
}
public static void ImprimeErrorMsje(String msje){
    JOptionPane.showMessageDialog(null,msje,"Salida",JOptionPane.INFORMATION_MESSAGE);
}
```

- Convertir un numero entero a factorial

  En el siguiente código iterativo, podemos apreciar que se declaran 2 variables, donde f servirá para almacenar el resultado de f*c, mientras que c, servirá para ser incrementado tantas veces el usuario lo desee.

```
public static double factorial(int dato)
{
    int f=1,c=1;
    while(c<=dato)
    {
        f*=c;
        c++;
    }return f;
}
```

Observamos que, en un ejercicio iterativo se hace uso de un while, como también podría ser do while. En este caso, el código se ejecutará mientras la condición se cumpla.

```
    while(c<=dato)
    {
        f*=c;
        c++;
    }return f;
}
```

El resultado satisfactorio de la ejecución del código, en el caso de que el usuario desee calcular la factorial de 5, será 120, como se muestra en la consola.

```
package Metodos;
import Metodos.FuncionesIterativas;

public class Test {
public static void main(String[]args) {
    System.out.println("Factorial: "
    +FuncionesIterativas.factorial(5));
```

```
<terminated> Test (1) [Java
Factorial: 120.0
```

Ahora, el código es realizado como un ejercicio recursivo, para el cual, utilizaremos dos códigos que, en complemento obtendrán la conversión de valores enteros a factoriales. El primer código será el encargado de realizar las operaciones correspondientes al problema:

```
public static double factorialRecur(int dato, int c) {
    if(dato==0||dato==1) return 1;
    else
        if(c<=dato)
        return c*factorialRecur(dato,c+1);
        else return 1;
}
```

Podemos analizar que, en el código recursivo, la condicionante está dada utilizando únicamente **if**, algo que lo diferencia de los códigos en iterativo. Se

puede apreciar la diferencia principal entre los dos códigos: un ejercicio recursivo hará el llamado a si mismo mientras la condicionante se cumpla.

```
        if(c<=dato)
        return c*factorialRecur(dato,c+1);
        else return 1;
}
```

Para poder imprimir el resultado final, se realiza un segundo código recursivo que almacena los datos ya procesados y los imprime en un solo cuadro de salida. El siguiente código, también hace un llamado a sí mismo, además de llamar al código anteriormente explicado para trabajar en conjunto.

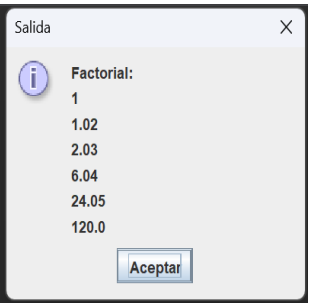
```
public static String listadoFac(int k)
{
    if(k<=15)
    {
        return k+"\n"+factorialRecur(k,1)+listadoFac(k+1);
    }
    else return "";

}
```

El resultado de la conversión puede apreciarse con el siguiente código. La cantidad de valores factoriales puede variar de acuerdo a lo que el usuario desee.

```
package Metodos;
import EntradaSalida.tools;

public class Test {
public static void main(String[]args) {
    tools.ImprimeMsje("Factorial: \n"+FuncionesRecursivas.listadoFac(1));
```



CONCLUSIONES

Terminando la práctica, podemos afirmar que la recursividad es una técnica que podemos implementar en la programación haciendo que un método se llame asi mismo, esto siempre y cuando se cumpla la condición para hacerlo. Es importante tener en cuenta que un programa recursivo nos puede facilitar la vida, como también nos la puede complicar si no se sabe como utilizar correctamente, puesto que, un programa podría desbordarse si la condición no esta bien especificada, lo cual crearía errores al momento de ejecutar el programa. Mientras que un ejercicio iterativo utiliza condicionantes como while o do-while, un ejercicio iterativo solo utiliza if como condicionante, esto para hacerlo mas practico y pequeño, en comparación con un ejercicio iterativo.

BIBLIOGRAFIA

- Marines M. 29/oct/2019. Diferencia entre iteración y recursividad. https://codigofacilito.com/articulos/articulo_16_10_2019_16_22_35
- Anónimo. S.f. 18.6. Algoritmos recursivos y algoritmos iterativos. https://uniwebsidad.com/libros/algoritmos-python/capitulo-18/algoritmos-recursivos-y-algoritmos-iterativos
- KeepCoding. 6/ene/2023. Que es la estructura iterativa en programación? https://uniwebsidad.com/libros/algoritmos-python/capitulo-18/algoritmos-recursivos-y-algoritmos-iterativos
- Universidad de Malaga. S.f. Tema 3. Recursividad. http://www.lcc.uma.es/~jlleivao/algoritmos/tt3.pdf#:~:text=Definici%C3%B3n%20de%20Recursividad%3A%20T%C3%A9cnica%20de%20programaci%C3%B3n%20muy%20potente,la%20invocaci%C3%B3n%20de%20un%20algoritmo%20a%20s%C3%AD%20mismo.
- Anónimo. S.f. Recursividad. EcuRed. https://www.ecured.cu/Recursividad